REMARKS

Claims 1-35 are pending in the present application. Claims 1-3, 4-19, and 21-35 were amended to recite "dynamically created installation property." Support for these amendments may be found on page 26, lines 15-18, of the specification. Reconsideration of the claims is respectfully requested.

Amendments were made to the Abstract as requested by the Examiner to conform to word limit guidelines. No new matter was entered as a result of these amendments.

I. 35 U.S.C. § 102, Anticipation, claims 1, 2, 5, 6, 11-14, 17, 18, 21, 22, 27-30, and 33-35

The Examiner has rejected claims 1, 2, 5, 6, 11-14, 17, 18, 21, 22, 27-30, and 33-35 under 35 U.S.C. § 102 (e) as being anticipated by *Gazdik et al* (U.S. Patent No. 6,301,708). This rejection is respectfully traversed.

With regard to claim 1 being anticipated by *Gazdik*, the Examiner states:

> Per claim 1:
> Gazdik discloses
> - a method of installation a program in a computing device ("A new method for installing . . . software . . ." in col. 3 line 29)
> - initiating installation of a program on the computing device ("An installer processing engine . . . runs on a computer system . . ." in col. 4 lines 3-4)
> - for each installation property used during installation of the program, storing the installation property in a fileset, and installing the fileset ("Each software component has associated therewith a unique component-specific data file . . . Each such data file co ntains the characteristics of the software component, as well as the commands to be executed for installation and unexecuted during uninstallation" in col. 3 lines 33-40)
>
> substantially as claimed.

Office Action, dated February 23, 2005, page 3.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir.1990). The *Gazdik* reference cited by the Examiner does not anticipate the present invention as recited in claim 1, because *Gazdik* fails to teach each and every element of the claim.

Independent claim 1, which is representative of independent claims 11, 13, 17, 27, 29, and 33-35 with regard to similarly recited subject matter, reads as follows:

> 1. A method of installing a program in a computing device, comprising:
> initiating installation of a program on the computing device;
> for each dynamically created installation property used during installation of the program, storing the dynamically created installation property in a fileset; and
> installing the fileset.

The rejected independent claim 1 recites, "for each dynamically created installation property used during installation of the program, storing the dynamically created installation property in a fileset." The feature of "storing the dynamically created installation property in a file set," as defined in claim 1, is not taught by *Gazdik*. As defined on page 26, lines 15-18 of the current specification, "Installation properties are any variables and variable values input by the user during installation or generated by the installation program during installation of the application."

Gazdik teaches a method for installing software that fragments a single script installation process into multiple independent files, based upon data files that are independent of process-control files. The first paragraph in the detailed description of *Gazdik* demonstrates *Gazdik*'s basic operation:

> A new method for installing and uninstalling software is provided which fragments the process so that the installation or uninstallation of each component of a software suite is controlled by multiple independent files, rather than by a single script file. Each software component has associated therewith a unique component-specific data file which is independent of and external to the installer-processing engine, the suite installation process flow, and the other software components. Each such data file contains the characteristics of the software component, as well as the commands to be executed for installation and unexecuted during uninstallation. The flow of the installation/uninstallation process is controlled by a separate process-control file which is read and executed by the install/uninstall processing engine, which for a preferred embodiment of the invention, is a state machine.

Gazdik, column 3, lines 29-44. Therefore, *Gazdik* simply fragments an installation process into independent files.

The Examiner alleges that *Gazdik* teaches the feature of storing the dynamically created installation properties, in the following cited section:

> Each software component has associated therewith a unique component-specific data file which is independent of and external to the installer-processing engine, the suite installation process flow, and the other software components. Each such data file contains the characteristics of the software component, as well as the commands to be executed for installation and unexecuted during uninstallation.

Gazdik, column 3, lines 33-40.

In the section above, *Gazdik* teaches data files that contain the characteristics of software components to be installed, as well as the commands to be executed for installation. Each of these data files contains pre-specified characteristics of the software component to be installed, and the scripted commands required to install the component, as further described in the following passage:

> **Each component persistent data file contains the characteristics of the software component, as well as the commands to be executed for installation and unexecuted during uninstallation.** This process is represented diagrammatically in FIG. 1. An installer processing engine 11, which includes a state machine 11A, runs on a computer system 14. The process flow of a state machine 11 is controlled by a process control state file 12. The process control state file 12 directs the state machine 11 to call on the component persistent data files 13A, 13B. . . 13n in an ordered sequence. **Each of the persistent data files 13A-13B provides the state machine 11 with the information required to install/uninstall a single software component, which are identified as component 1, component 2, and component n.** This information is used to control a computer system 14, which installs a selected collection of uninstalled program files belonging to a software suite, which may include either local uninstalled program files 15L or remote uninstalled program files 15R, on a connected mass storage device 16. **Because installation information for a specific software component resides in a component persistent data file, the manner in which a software component is installed can be changed with relative ease and rapidity, without affecting the install flow or how other software components are installed.** (emphasis added)

Gazdik, column 3, line 63, to column 4, line 22.

However, neither of the above sections, nor any other section in *Gazdik,* teach storing the dynamically created installation property in a fileset, wherein the dynamically created installation property is a variable and variable values input by the user during installation or generated by the installation program during installation. But that is not to say that *Gazdik* installations operate independent of variables and variable values input by users.

> The state machine transitions from that state object to another by evaluating the return code. Each state object has a specific task. For example, it may prompt the end user to provide a directory name and location; it may ask the user to identify those software components which he desired to install; or it may make one of many other install/uninstall-related queries.

Gazdik, column 5, lines 9-16.

Although *Gazdik* queries the user for a directory name and to identify those software components desired to be installed, nowhere in *Gazdik* is there any teaching or even a suggestion that these variables and variable values input by users are stored or saved in any manner. *Gadzik* does not mention or disclose storing dynamically created installation properties used during installation of the program in a fileset.

In contrast to *Gazdik,* claim 1 of the present invention includes the feature of storing dynamically created installation properties, defined as variables and variable values input by users or generated by the installation program during installation of the application. In *Gazdik,* the directory name and identity of software components to be installed input by the user are not stored in any storage device. *Gazdik* teaches information required for installing a software component, but *Gazdik* does not store such information for use by subsequent installations or uninstallations as recited in claim 1. In other words, *Gazdik* does not teach the feature of storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application.

Therefore, *Gazdik* fails to teach all elements of the claimed invention in claim 1. The other independent claims include similar features not taught by the cited reference.

Thus, *Gazdik* fails to anticipate the present invention as recited in independent claims 1, 11, 13, 17, 27, 29, and 33-35.

Furthermore, *Gazdik* does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. Absent the Examiner pointing out some teaching or incentive to implement *Gazdik* storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, one of ordinary skill in the art would not be led to modify *Gazdik* to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify *Gazdik* in this manner, the presently claimed invention can be reached only through an improper use of hindsight using the Applicants' disclosure as a template to make the necessary changes to reach the invention.

Therefore the rejection of independent claims 1, 11, 13, 17, 27, 29, and 33-35 under 35 U.S.C. §102 has been overcome.

Claims 2-10, 12, 14-16, 18-26, 28, and 30-32 are dependent claims depending on independent claims 1, 11, 13, 17, 27, and 29, respectively. Claims 2-10, 12, 14-16, 18-26, 28, and 30-32 are also allowable, at least by virtue of their dependency on allowable claims. Further, these dependent claims include other features not taught or disclosed by *Gadzik*. For example, claim 2, which is representative of claim 18 with regard to similarly recited subject matter, reads as follows:

> 2. The method of claim 1, further comprising:
> generating a registry object for each dynamically created installation property: and
> storing the registry object in a system product registry for the program.

Because *Gazdik* does not teach storing a dynamically created installation property in a fileset, wherein the dynamically created installation property is a variable and variable values input by the user during installation or generated by the installation program during installation, *Gazdik* also does not teach "generating a registry object for each dynamically created installation property." Therefore, because *Gazdik* does not teach generating a registry object for each dynamically created installation property, *Gazdik* has no reason to teach "storing the registry object in a system product registry for the program."

Thus, the rejection of claims 1, 2, 5, 6, 11-14, 17, 18, 21, 22, 27-30, and 33-35 under 35 U.S.C. §102 has been overcome.

II. <u>35 U.S.C. § 103, Obviousness, claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32</u>

The Examiner has rejected claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32 under 35 U.S.C. §103(a) as being unpatentable over *Gazdik et al*, U.S. Patent No. 6,301,708, ("*Gadzik*") in view of *Cassady-Dorion et al.*, "Industrial Strength Java" ("*Cassady-Dorion*") This rejection is respectfully traversed.

The Examiner bears the burden of establishing a prima facie case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). For an invention to be prima facie obvious, the prior art must teach or suggest all claim limitations. *In re Royka,* 490 F.2d 981, 180 USPQ 580 (CCPA 1974). *Gazdik* and *Cassady-Dorion* do not render obvious the present invention as recited in claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32 because the references fail to teach or suggest all claim limitations found in the independent claims. Further, *Cassady-Dorion* alone or in combination with *Gazdik* does not teach or suggest the features in the independent claims.

Claims 3-4 and 7-8 are dependent upon independent claim 1, claims 15-16 are dependent upon independent claim 13, claims 19-20 and 23-24 are dependent upon independent claim 17, and claims 31-32 are dependent upon independent claim 29. As argued in response to the rejections of claims 1, 11, 13, 17, 27, 29, and 33-35 above, *Gazdik* does not teach the feature of storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application.

Cassady-Dorion also does not teach this feature. The sections in *Cassady-Dorion* provided by the Examiner do not teach the features of storing dynamically created installation properties. The Examiner mentioned the following sections of *Cassady-Dorion*:

> "You can store large information sets in many ways, but the hash table data structure is one of the most popular" on page 210, section titled 'Storing Information in a Hash Table.' . . ." The serialization services take a data structure as input . . . and produce an encoded stream of bytes as output. This serial stream can then be written . . . to storage media, such as a hard drive" on page 472, section titled 'Serializing Objects.'"

Office Action, dated February 23, 2005, pages 8-9.

In the section referenced above that begins on page 210, *Cassady-Dorion* teaches the "need to have constant access to various parts of some information set," such that "speed becomes an important issue." Therefore, "a hash table data structure is one of the most popular" to "store large information sets" because it "works by dividing the large data set into small groups," and each "of these small groups is stored sequentially." *Cassady-Dorion* describes the explicit title of the section, storing information in a hash table. *Cassady-Dorion*, however, does not provide any teaching, suggestion, or incentive for storing each dynamically created installation property used during installation of the program in a fileset. While a hash table is related to a fileset, a hash table is not a fileset, as demonstrated by claim 3 of the present invention, which includes "storing the dynamically created installation property in a <u>hashtable of the fileset</u>." (emphasis added) *Cassady-Dorion* is directed towards the general use of hash tables without any teaching or suggestion to storing the specific data recited in claim 1. Although this cited reference may teach storing data, no teaching or suggestion of storing the type of data recited in claim 1 is taught or suggested by this reference

In the section referenced above that begins on page 472, *Cassady-Dorion* teaches that "serialization services take a data structure as input (such as an object or group of objects) and produce an encoded stream of bytes as output. This serial stream can then be written across a network or modem connection, or to storage media, such as a hard drive." *Cassady-Dorion* describes the explicit title of the section, serializing objects.

Storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, is not discussed in *Cassady-Dorion*. Furthermore, *Cassady-Dorion* does not teach anything about installations or dynamically created installation properties, much less storing dynamically created installation properties. The serialization of an object can be applied to any data structure, and any data set can be stored as information in a hash table. But general procedures that can be applied to any data set or data structure are not the same as, or even remotely suggest, a feature of storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application. Therefore, *Cassady-Dorion* does not compensate for the deficiencies in the teachings of *Gazdik*.

The Examiner further states that it would be obvious to combine *Gazdik* with *Cassady-Dorion* to arrive at the features of the claimed invention. One of ordinary skill in the art would not combine *Gazdik* with *Cassady-Dorion* when each reference is considered as a whole. In considering a reference as a whole, one of ordinary skill in the art would take into account the problems recognized and solved. As discussed above, *Gazdik* is directed towards information required for installing a software component. (*Gazdik*, Abstract). To achieve this purpose, in *Gazdik*, fragments files with this information. In contrast, *Cassady-Dorion* is directed towards storing information in a hash table and serializing objects. Thus, *Cassady-Dorion* is a method for the sending and storing of information, and not directed towards addressing the same problems as methods to install and uninstall software.

In addition, there is no motivation to combine the teachings of *Gazdik* with *Cassady-Dorion* in the manner alleged by the Examiner. The Examiner alleges that the motivation for the alleged combination is "it would allow a user to easily locate a specific part of a data set." (*Office Action*, dated February 23, 2005, page 9). The mere fact that a prior art reference can be readily modified does not make the modification obvious unless the prior art suggested the desirability of the modification. *In re Laskowski*, 871 F.2d 115, 10 U.S.P.Q.2d 1397 (Fed. Cir. 1989) and also *see In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992) and *In re Mills*, 916 F.2d 680, 16 U.S.P.Q.2d 1430 (Fed. Cir. 1993).

Gazdik has no suggestion of a need for storing dynamically created installation properties, or variables and variable values input by users during installation or generated by the installation program during installation of the application. In fact, *Gazdik* is directed toward fragmenting files with information required for installing a software component. There is no need, let alone any suggestion, in *Gazdik* to store dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application. Moreover, there is no suggestion in *Cassady-Dorion* of a need to integrate the storing and sending of information approach of *Cassady-Dorion* with the information required for installing a software component approach, such as that taught by *Gazdik*. *Cassady-Dorion* has nothing to do with installation of software components. There is no need, let alone any suggestion in *Cassady-Dorion* to provide information required for installation of software components. Thus, the alleged motivation offered by the Examiner is not based on the actual teaching of the references.

Even if *Gazdik* were combinable with *Cassady-Dorion*, the result of such a combination would not be the invention as recited in independent claim 1. Rather, such an alleged combination would result in a system for fragmenting information files required for installing software components, substantially as taught in *Gazdik*, with the sending and storing of information, in the manner described by *Cassady-Dorion*. As noted above, the Examiner quotes *Cassady-Dorion* to state that "You can store large information sets in many ways, but the hash table data structure is one of the most popular." Combining *Cassady-Dorion* with *Gazdik* would not result in storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, unless such a combination used the application for the present invention as a template, because many possibilities exist for storing numerous types of *Gazdik* information using the *Cassady-Dorion* hash table data structure that differ from the storing of variables and variable values input by users. Additionally, the same point is true in regards to serializing object as taught by *Cassady-Dorion*. As noted above, the Examiner quotes *Cassady-Dorion* to state that "serial stream can then be written . . . to storage media, such as a hard drive." Combining *Cassady-Dorion* with *Gazdik* would not result in serializing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, unless such a combination used the present invention as a template, because many possibilities exist for serializing numerous types of *Gazdik* information using the *Cassady-Dorion* serialization method that differs from the serializing of variables and variable values input by users during installation or generated by the installation program during installation of the application. As noted in the survey of the prior art, and the discussion above regarding the lack of a teaching or even a suggestion in *Gazdik* to serialize dynamically created installation properties as defined by the present invention, the serializing of variables and variable values input by users during installation or generated by the installation program during installation of the application is not obvious. Even with the alleged additions of *Gazdik* and *Cassady-Dorion*, it would not be obvious to serialize and store dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, as recited in claim 1 of the present invention.

Thus, the combination of *Gazdik* and *Cassady-Dorion* fail to teach or suggest the present invention as cited in claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32. Although *Cassady-Dorion* may teach storing information in a hash table and serializing objects, neither *Cassady-Dorion* nor *Gazdik*, either alone or in combination, teach or suggest all the claim limitations in claims 3-4 and 7-8, (dependent upon claim 1) or in claims 15-16, (dependent upon claim 13), or in claims 19-20 and 23-24, (dependent upon claim 17), or in claims 31-32, (dependent upon claim 29), as argued in response to the rejections of claims 1, 11, 13, 17, 27, 29, and 33-35 above. The features relied upon as being taught in *Gazdik*, the features of storing dynamically created installation properties, variables and variable values input by users during installation or generated by the installation program during installation of the application, are not taught or suggested by that reference, as explained above. As a result, a combination of these references would not reach the claimed invention in claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32.

Thus, the rejection of claims 3, 4, 7, 8, 15, 16, 19, 20, 23, 24, 31 and 32 under § 103 over *Gazdik* in view of *Cassady-Dorion* has been overcome.

III. Conclusion

It is respectfully urged that the subject application is patentable over the cited references and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: 5/23/05

Respectfully submitted,

[illegible]

Cathrine K. Kinslow
Reg. No. 51,886
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicant

CK/jl